

**Japanese Patent Office
Patent Laying-Open Gazette**

Patent Laying-Open No. 11-312152
Date of Laying-Open: November 9, 1999
International Class(es): G 06 F 15/16
H 04 Q 7/38

(28 pages in all)

Title of the Invention: Mobile Electronic Device and Controlling Method Thereof

Patent Appln. No. 10-365666
Filing Date: December 22, 1997
Priority Claimed: Country: U.S.A.
Filing Date: December 22, 1997
Serial No. 995606

Inventor(s): Michael McMahon
Marion C. Lineberry

Applicant(s): Texas Instruments Inc.

(transliterated, therefore the spelling might be incorrect)

THIS PAGE BLANK (USPTO)

(19)日本国特許庁 (JP)

(12) 公開特許公報 (A)

(11)特許出願公開番号

特開平11-312152

(43)公開日 平成11年(1999)11月9日

(51) Int.Cl.
G 06 F 15/16
H 04 Q 7/38識別記号
620F I
G 06 F 15/16
H 04 B 7/26620 G
109 M

審査請求 未請求 請求項の数2 OL (全28頁)

(21)出願番号 特願平10-365666

(22)出願日 平成10年(1998)12月22日

(31)優先権主張番号 995606

(32)優先日 1997年12月22日

(33)優先権主張国 米国(US)

(71)出願人 590000879
 テキサス インスツルメンツ インコーポ
 レイテッド
 アメリカ合衆国テキサス州ダラス、ノース
 セントラルエクスプレスウェイ 13500
 (72)発明者 マイケル マクマホン
 アメリカ合衆国 テキサス州プラノ、メリ
 アン ドライブ 3817
 (72)発明者 マリオン シー. ラインパリイ
 アメリカ合衆国 テキサス州ダラス、エ
 ヌ. セルバ ドライブ 1250
 (74)代理人 弁理士 浅村 鮎 (外3名)

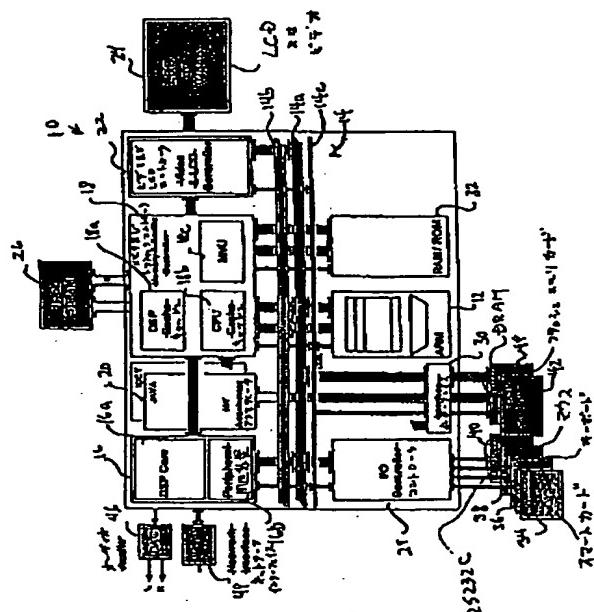
最終頁に続く

(54)【発明の名称】 移動電子装置および制御方法

(57)【要約】

【課題】 多数のプロセッサおよびコプロセッサの使用をグレードアップし最適化するデータ処理アーキテクチャを得る。

【解決手段】 複数個のプロセッサ12, 16を含むワイヤレスデータプラットホーム10。タスクが実施される時に情報を通信できるようにプロセッサ間に通信チャネルが開設される。1個のプロセッサで実施されるダイナミッククロスコンバイラー80がコードを別のプロセッサのためのネーティブ処理コードへコンパイルする。ダイナミッククロスリンカー82が他のプロセッサのためにコンパイルされたコードをリンクする。ネーティブコードはそれを包むJAVA Bean90(もしくは他の言語タイプ)を使用してプラットホームへダウンロードすることもできる。JAVA Beanは暗号化しセキュリティのためにデジタルサインすることができる。



【特許請求の範囲】

【請求項1】 移動電子装置であって、
ネイティブコードを実行するコプロセッサと、
ホストプロセッサシステムおよびプロセッサ独立コード
に対応するネイティブコードを実行するように作動する
ホストプロセッサシステムであって、デジタル信号コプロセッサにより実施されるタスクをダイナミックに変化
せるように作動するホストプロセッサシステムと、
前記ホストプロセッサシステムと前記コプロセッサ間の
通信回路と、
を含む移動電子装置。

【請求項2】 移動電子装置の制御方法であって、
コプロセッサにおいてネイティブコードを実行するステップと、
ホストプロセッサシステムにおいてネイティブコードおよびプロセッサ独立コードを実行するステップと、
デジタル信号コプロセッサにより実施されるタスクを前記ホストプロセッサシステムによりダイナミックに変える
ステップと、
前記ホストプロセッサシステムと前記コプロセッサ間で
通信を行うステップと、
を含む方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】 本発明は一般的に移動電子装置に関し、特に移動電子装置用ハードウェアおよびソフトウェアプラットホームに関する。

【0002】

【従来の技術】 装置のパワー、したがって機能、が増強されるにつれてハンドヘルド携帯装置の人気が高まっている。近年PDA (Personal Digital Assistant) が広く使用されており、セルラー電話機とPDAのいくつかの能力を組み合わせたスマートホーン (Smart phones) が近未来の通信に著しいインパクトを与えるものと考えられる。

【0003】 現在ある装置には、音声認識等の、ある個別の機能を提供する1個以上のDSP (デジタル信号処理装置) もしくは他のコプロセッサ、および他のデータ処理機能のための汎用プロセッサが内蔵されている。DSP用コードおよび汎用プロセッサ用コードは一般的にROMその他の非揮発性メモリに格納され、それらは容易に修正することができない。したがって、改善および新しい機能を利用できるようになった時に、しばしば装置の能力をグレードアップできないことがある。特に、装置内に存在することがあるDSPや他のコプロセッサを最大限に使用することができない。

【0004】

【発明が解決しようとする課題】 したがって、多数のプロセッサおよびコプロセッサの使用をグレードアップし最適化することができるデータ処理アーキテクチャが

必要とされている。

【0005】

【課題を解決するための手段】 本発明において、移動電子装置はネイティブコードを実行するコプロセッサと、ホストプロセッサシステムに対応するネイティブコードおよびプロセッサ独立コードを実行するように作動するホストプロセッサシステムとを含んでいる。ホストプロセッサシステムはデジタル信号コプロセッサにより実施されるタスクをダイナミックに変えるように作動する。通信回路によりホストプロセッサシステムとコプロセッサ間の通信が提供される。

【0006】 本発明により従来技術を凌ぐ著しい利点が提供される。それは、ホストプロセッサシステムが、デジタル信号処理装置であっても良いコプロセッサにより実施されるタスクをダイナミックに割り当ててコプロセッサを完全に使用できるためである。ホストプロセッサシステムは、各プロセッサの現在の能力等の、多様な要因に応じて複数のコプロセッサの1つへ直接ルーチンすることができる。

【0007】

【発明の実施の形態】 図1に例えばスマートホーンやPDAを実施するのに使用できる一般的なワイヤレスデータプラットホームアーキテクチャの好ましい実施例を示す。ワイヤレスデータプラットホーム10は、データバス14a、アドレスバス14bおよびコントロールバス14cを含むバス構造14に接続された汎用(ホスト)プロセッサ12を含んでいる。コアプロセッサ16aおよび周辺インターフェイス16bを含む1個以上のDSP (もしくは他のコプロセッサ) 16がバス14およびDSPキャッシュメモリ18a、CPUキャッシュ18b、およびMMU (メモリマネジメントユニット) 18cを含むメモリおよびトラフィックコントローラ18に接続されている。ハードウェアアクセラレータ回路20 (JAVA等のポータブル言語を加速する) およびビデオおよびLCDコントローラ22もメモリおよびトラフィックコントローラ18に接続されている。ビデオおよびLCDコントローラの出力はLCDもしくはビデオディスプレイ24に接続されている。

【0008】 メモリおよびトラフィックコントローラ18はバス14およびSDRAM (同期ダイナミックランダムアクセスメモリ) として示す、主記憶装置26に接続されている。バス14はI/Oコントローラ28、インターフェイス30、およびRAM/ROM32にも接続されている。複数の装置をスマートカード34、キーボード36、マウス38等のワイヤレスデータプラットホーム10もしくはUSB (ユニバーサルシリアルバス) ポートもしくはRS232シリアルポート等の1つ以上のシリアルポート40に接続することができる。インターフェイス30はフラッシュメモリカード42および/もしくはDRAMカード44に接続することができ

る。周辺インターフェイス16bはDSP16をDAC（デジタル／アナログコンバータ）46、ネットワークインターフェイス48もしくは他の装置に接続することができる。

【0009】図1のワイヤレスデータプラットホーム10は汎用プロセッサ12およびDSP16を利用してい。DSP16が特定の固定機能専用とされている現在の装置とは異なり、図1のDSP16は任意数の機能を使用することができる。それにより、ユーザはDSP16の利益を完全に引き出すことができる。

【0010】DSP16を使用できる1つの主要なエリアはマンマシンインターフェイス(MMI)に関連している。重要なのは、音声認識、イメージおよびビデオ圧縮、データ暗号化、テキスト-音声変換、等の機能をDSP16を使用してより効率的に実施できることである。本アーキテクチャにより新しい機能や改善をワイヤレスデータプラットホーム10へ容易に付加することができる。

【0011】ワイヤレスデータプラットホーム10は一般的なブロック図であり、さまざまに修正することができる。例えば、図1には独立したDSPおよびプロセッサキャッシュ18a、18bが図示されている。当業者にはお判りのように、ユニット型キャッシュも使用することができる。さらに、ハードウェアアクセラレーション(acceleration)回路20はオプショナルアイテムである。このような装置によりJAVA等の言語の実行が加速されるが、この回路は装置の動作に必要なものではない。さらに、1個のDSPしか図示されていないが、多数のDSP（もしくは他のコプロセッサ）をバスに接続することができる。

【0012】図2にワイヤレスデータプラットホーム10の機能的ソフトウェアアーキテクチャを示す。このブロック図はJAVAを使用するものと想定しており、JAVA以外の言語も使用することができる。機能的に、ソフトウェアは2つのグループへ分割され、それはホストプロセッサソフトウェアおよびDSPソフトウェアである。ホストソフトウェアは1つ以上のアプレット40を含んでいる。DSP APIクラス42はJAVAアプリケーション用JAVA APIパッケージ、すなわちDSP API50およびホストDSPインターフェイスレイヤ52の機能へアクセスするアプレットである。JAVAバーチャルマシン(VM)44がアプレットを解釈する。JAVAネイティブインターフェイス46は、JAVA VMがホストプロセッサもしくは特定プラットホームコードを実行する方法である。ネイティブタスク48はJAVAネイティブインターフェイスを使用せずにホストプロセッサ12により実行することができる非JAVAプログラムである。後述するDSP

API50は、DSP16の能力を使用するためにコールするホスト12で使用されるAPI（アプリケーシ

ョンプログラムインターフェイス）である。ホスト-DSPインターフェイスレイヤ52は他のタスク、あるいはホスト-DSP通信プロトコルを介したチャネルを使用して他のハードウェアと互いに通信するためのAPIをホスト12およびDSP16に提供する。DSPデバイスドライバ54はDSP16と通信するためのホストRTOS56（リアルタイムオペレーティングシステム）用ホストベースデバイスドライバである。ホストRTOS56はアクセラレーテッドテクノロジー社のNUCLEUS PLUS等のオペレーティングシステムである。マイクロソフト社のWINDOWSCE等の非リアルタイムオペレーティングシステムを使用することもできる。DSPライブラリ58にはDSP16で実行するために格納されたプログラムが含まれている。

【0013】DSP側で、DSP16により実行する1つ以上のタスク60をメモリ内に格納することができる。後述するように、DSPの機能がスタティックではなくダイナミックとなるように、タスクを所望によりメモリから出し入れすることができる。DSP側のホスト-DSPインターフェイスレイヤ62はホスト側のホスト-DSPインターフェイスレイヤ52と同じ機能を実施する、すなわちホスト12およびDSP16は通信することができる。DSP RTOS64はDSPプロセッサ用オペレーティングシステムである。ホストデバイスドライバ66はホスト12と通信するDSP RTOS64用DSPベースデバイスドライバである。ホスト-DSPインターフェイス70はDSP16をホスト12に接続する。

【0014】動作において、図2に示すソフトウェアアーキテクチャはDSP16を従来技術のように固定機能装置ではなく、可変機能装置として使用する。したがって、DSP機能は図2のアーキテクチャを内蔵する移動装置へダウンロードして、DSP16がホスト12に対してさまざまな信号処理機能を実施できるようにすることができる。

【0015】DSP-API

DSP-APIはホスト12からDSP16へのデバイス独立インターフェイスを提供する。この機能により、ホスト12にはDSP16にタスクをロードしてスケジュールを行いかつこれらのタスクをコントロールして通信する能力が与えられる。API機能には、DSPの利用可能な資源を決定し、ホスト12およびDSPタスクを生成かつコントロールし、ホスト12およびDSPタスク間のデータチャネルを生成かつコントロールし、タスクと通信するための呼出しが含まれる。これらの機能については後述する。各機能はブール結果を戻し、それは成功したオペレーションに対するSUCCESS、もしくはFAILUREである。結果がFAILUREであれば、errcodeをチェックしてどのエラーが発生したかを確認しなければならない。

【0016】

DSP_Get_MIPS

```
BOOL DSP_Get_MIPS(T_DeviceID DevID, U32 *mips, UI6 *errcode);
```

この関数はDSP上で利用できる現在のMIPSを戻す。それはDSP16のMIPS能力マイナースペースMIPS値(付加ダイナミックタスクの無いMIPS値、すなわちカーネルプラスAPIコードプラスドライバ)、マイナスロードされた全てのダイナミックタスクに対するMIPSレーティングの和である。errcodeパラメータは下記の考えられる結果を含む。

DSP_Get_Memory_Available

```
BOOL DSP_Get_Memory_Available(T_DeviceID DevID, T_Size *progmem, T_Size *datamem, UI6 *errcode);
```

この関数はプログラムメモリおよびデータメモリの両方に対して利用できるメモリ量についてDevIDにより指定されるDSP16に質問する。その結果得られる値はprogmemおよびdatamemパラメータ内に戻される。サイズはT_DSP_Words内に指定される。errcodeパラメータは下記の考えられる結果を含む。

DSP_Alloc_Mem

```
BOOL DSP_Alloc_Mem(T_DeviceID DevID, UI6 mempage, T_Size size, T_DSP_Word **memptr, UI6 *errcode);
```

この関数によりDSP16上に1ブロックのメモリが割り当てられる。DevIDはどのデバイス上にメモリを割り当てるかを指定する。mempageはプログラムスペースに対しては0であり、データスペースに対しては1である。sizeパラメータはT_DSP_Words内のメモリブロックサイズを指定する。戻されるmemptrはDSP16上のメモリブロックへのポインター、もしくは失敗時のNULLである。errcodeパラメータは下記の考えられる結果を含む。

DSP_Free_Mem

```
BOOL DSP_Free_Mem(T_DeviceID DevID, UI6 mempage, T_DSP_Word *memptr, UI6 *errcode);
```

この関数はDSP

Allc_Mem関数を割り当てられたDSP上の1ブロックのメモリを解放する。DevIDはどのデバイス上にメモリが常駐するかを指定する。mempageはプログラムスペースに対しては0であり、データスペースに対しては1である。memptrパラメータはメモリブロックへのポインターである。errcodeパラメータは下記の考えられる結果を含む。

DSP_Get_Code_Inf

```
BOOL DSP_Get_Code_Inf(char *Name, T_CodeHdr *codehdr, UI6 *errcode);
```

【数1】

【数2】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING

【0017】

【数3】

【数4】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING

【0018】

【数5】

【数6】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_MEMPAGE
DSP_NOT_ENOUGH_MEMORY

【0019】

【数7】

【数8】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_MEMPAGE
DSP_MEMBLOCK_NOT_FOUND

【0020】

【数9】

この関数はDSPライブラリテープルへアクセスし、Nameパラメータにより指定されるDSP関数コードのコードヘッダーを戻す。戻されると、codehdrパラメータにより指示される位置はコードヘッダー情報を含む。errcodeパラメータは下記の考えられる結果を含む。

DSP_Link_Code

```
BOOL DSP_Link_Code(T_DeviceID DevID, T_CodeHdr *codehdr,
                    T_TaskCreate *tcs, UI6 *errcode);
```

この関数はDevIDにより指定されるDSP上の指示されたアドレスで走るようにDSP関数コードをリンクする。codehdrパラメータは関数のコードヘッダーを指示する。ダイナミッククロスリンカーはコードヘッダー内およびコード(COFFファイル)内の情報を基づいてコードをリンクする。ダイナミッククロスリンカーは必要に応じてメモリを割り当て、DSP16にコードをリンクしロードする。tcsパラメータはDSP_Create_Task関数において必要なタスク生成構造へのポインターである。DSP_Link_Codeはタスクの生成に備えて構造のコードエントリーポイント、優先順位、およびクオントム(quantum)

DSP_Put_BLOB

```
BOOL DSP_Put_BLOB(T_DeviceID DevID, T_HostPtr srcaddr, T_DSP_Ptr
                   destaddr, UI6 mempage, T_Size size, UI6 *errcode);
```

この関数は指定されたBLOB(Binary Large Object)をDSP16へコピーする。DevIDはどのDSP16へオブジェクトをコピーするかを指定する。srcaddrパラメータはホストメモリ内のオブジェクトへのポインターである。destaddrはDSP16上のオブジェクトをコピーする位置へのポインターである。mempageはプログラムスペースに対しては0であり、データスペースに対しては1である。サイズパラメータはTDSP_Words内の

DSP_Create_Task

```
BOOL DSP_Create_Task(T_DeviceID DevID, T_TaskCreate *tcs, T_TaskID
                      *TaskID, UI6 *errcode);
```

DSP

Create_Taskは、タスクパラメータおよびDSPのプログラムスペース内のコード位置が与えられたらタスクを生成するようDSP16に要求する。タスク

【数10】

```
DSP_SUCCESS  
DSP_NAMED_FUNC_NOT_FOUND
```

【0021】

【数11】

m) フィールドを充填する。errcodeパラメータは下記の考えられる結果を含む。

【数12】

```
DSP_SUCCESS  
DSP_DEVID_NOT_FOUND  
DSP_DEVID_NOT_RESPONDING  
DSP_NOT_ENOUGH_PROG_MEMORY  
DSP_NOT_ENOUGH_DATA_MEMORY  
DSP_COULD_NOT_LOAD_CODE
```

【0022】

【数13】

オブジェクトのサイズを指定する。errcodeパラメータは下記の考えられる結果を含む。

【数14】

```
DSP_SUCCESS  
DSP_DEVID_NOT_FOUND  
DSP_DEVID_NOT_RESPONDING  
DSP_INVALID_MEMPAGE
```

【0023】

【数15】

生成構造を表1に示す。

【0024】

【表1】

表1 タスク生成構造

データタイプ	フィールド名	説明
T_DSP_Name	Name	User defined name for the task. (タスクのユーザ定義名)
U32	MIPS	MIPS used by the task. (タスクが使用するMIPS)
T_ChnlID	ChanIn	The channel ID used for task input. (タスク入力に使用するチャネルID)
T_ChnlID	ChanOut	The channel ID used for task output. (タスク出力に使用するチャネルID)
T_StmID	StmIn	The stream ID used for task input. (タスク入力に使用するストリームID)
T_StmID	StmOut	The stream ID used for task output. (タスク出力に使用するストリームID)
U16	Priority	The task's priority. (タスクの優先順位)
U32	Quantum	The task's timeslice in system ticks. (システムチック内のタスクのタイムスライス)
T_Size	StackReq	The amount of stack required. (所有タスク量)
T_DSP_Ptr	MsgHandler	Pointer to code to handle messages to the task. (タスクへのメッセージを処理するコードへのポインター)
T_HOST_Ptr	CallBack	Pointer to Host code to handle messages from the task. (タスクからのメッセージを処理するホストコードへのポインター)
T_DSP_Ptr	Create	Pointer to code to execute when task is created. (タスク生成時に実行するコードへのポインター)
T_DSP_Ptr	Start	Pointer to code to execute when task is started. (タスク開始時に実行するコードへのポインター)
T_DSP_Ptr	Suspend	Pointer to code to execute when task is suspended. (タスク保留時に実行するコードへのポインター)
T_DSP_Ptr	Resume	Pointer to code to execute when task is resumed. (タスク再開時に実行するコードへのポインター)
T_DSP_Ptr	Stop	Pointer to code to execute when task is stopped. (タスク停止時に実行するコードへのポインター)

【0025】タスクが生成されると、Createエントリーポイントが呼び出され、任意の必要な予備初期化を行う機会をタスクに与える。Create, Suspend, Resume, およびStopエントリーポイントはNULLとすることができる。その結果得られるTask IDはデバイスID(Dev ID)とDSPのタスクIDの両方を含んでいる。Task IDがNULLであれば、生成は失敗である。errcodeパラメータは下記の考えられる結果を含む。

DSP_Start_Task

```
BOOL DSP_Start_Task(T_TaskID TaskID, U16 *errcode);
```

この関数はTask IDにより指定されるDSPタスクを開始する。タスクのStartエントリーポイントで実行が開始する。errcodeパラメータは下記の考えられる結果を含む。

【数18】

```
DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND
```

【数16】

```
DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_PRIORITY
DSP_CHANNEL_NOT_FOUND
DSP_ALLOCATION_ERROR
```

【0026】

【数17】

【0027】

【数19】

DSP_Suspend_Task

```
BOOL DSP_Suspend_Task(T_TaskID TaskID, UI6 *errcode);
```

この関数は Task ID により指定された DSP タスクを保留する。保留される前に、任意の必要なハウスキーピングを実施する機会をタスクに与えるためにタスクの Suspend エントリーポイントが呼び出される。errcode パラメータは下記の考えられる結果を含む。

【数20】

DSP_Resume_Task

```
BOOL DSP_Resume_Task(T_TaskID TaskID, UI6 *errcode);
```

この関数は DSP Suspend_Task により保留された DSP タスクを再開する。再開される前に、任意の必要なハウスキーピングを実施する機会をタスクに与えるためにタスクの Resume エントリーポイントが呼び出される。errcode パラメータは下記の考えられる結果を含む。

【数22】

DSP_Delete_Task

```
BOOL DSP_Delete_Task(T_TaskID TaskID, UI6 *errcode);
```

この関数は Task ID により指定された DSP タスクを削除する。削除する前に、任意の必要なクリーンナップを実施する機会をタスクに与えるためにタスクの Stop エントリーポイントが呼び出される。それはタスクにより割り当てられた任意のメモリを解放し、タスクが取得した任意の資源を戻すことを含まなければならぬ。errcode パラメータは下記の考えられる結果を含む。

DSP_Change_Task_Priority

```
BOOL DSP_Change_Task_Priority(T_TaskID TaskID, UI6 newpriority, UI6 *oldpriority, UI6 *errcode);
```

この関数は Task ID により指定された DSP タスクの優先順位を変える。優先順位は newpriority へ変えられる。newpriority の考えられる値はRTOS 依存である。戻ると、oldpriority パラメータはタスクの前の優先順位に設定される。errcode パラメータは下記の考えられる結果を含む。

【数26】

DSP_Get_Task_Status

```
BOOL DSP_Get_Task_Status(T_TaskID TaskID, UI6 *status, UI6 *priority, T_ChanID *Input, T_ChanID Output, UI6 *errcode);
```

この関数は Task ID により指定された DSP タスクの状態を戻す。status は下記の値の 1 つを含む。

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND

【0028】
【数21】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND
DSP_TASK_NOT_SUSPENDED

【0029】
【数23】

【数24】
DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND

【0030】
【数25】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND
DSP_INVALID_PRIORITY

【0031】
【数27】

【数28】

DSP_TASK_RUNNING
DSP_TASK_SUSPENDED
DSP_TASK_WAITFOR_SEM
DSP_TASK_WAITFOR_QUEUE
DSP_TASK_WAITFOR_MSG

【0032】 priorityパラメータはタスクの優先順位を含み、InputおよびOutputパラメータはタスクの、それぞれ、入力および出力IDを含む。
 errcodeパラメータは下記の考えられる結果を含む。

DSP_Get_ID_From_Name

```
BOOL DSP_Get_ID_From_Name(T_DeviceID DevID, T_DSP_Name Name,
                           T_DSP_ID *ID, UI6 *errcode);
```

この関数はDSP16上の名前を付けたオブジェクトのIDを戻す。名前を付けたオブジェクトはチャネル、タスク、メモリロック、もしくは任意他のサポートされた名前を付けたDSPオブジェクトとすることができる。errcodeパラメータは下記の考えられる結果を含む。

【数31】

DSP_Dbg_Read_Mem

```
BOOL DSP_Dbg_Read_Mem(DEVICE_ID DevID, U8 mempage, DSP_PTR
                       addr, U32 count, DSP_WORD *buf, UI6 *errcode);
```

この関数はメモリの1ブロックを要求する。mempageはプログラムメモリ(0)もしくはデータメモリ(1)を指定する。addrパラメータはメモリ開始アドレスを指定し、countはどれだけ多くのT_SD_P_Wordsを読み出すかを示す。bufパラメータはメモリをコピーしなければならないバッファーを提供了呼び出し者へのポインターである。errcodeパラメータは下記の考えられる結果を含む。

DSP_Dbg_Write_Mem

```
BOOL DSP_Dbg_Write_Mem(T_DeviceID DevID, UI6 mempage, T_DSP_Ptr
                        addr, T_Count count, T_DSP_Word *buf, UI6 *errcode);
```

この関数はメモリの1ブロックを書き込む。mempageはプログラムメモリ(0)もしくはデータメモリ(1)を指定する。addrパラメータはメモリ開始アドレスを指定し、countはどれだけ多くのT_SD_P_Wordsを書き込むかを示す。bufパラメータは書き込むメモリを含むバッファーへのポインターである。errcodeパラメータは下記の考えられる結果を含む。

DSP_Dbg_Read_Reg

```
BOOL DSP_Dbg_Read_Reg(T_DeviceID DevID, UI6 RegID, T_DSP_Word
                       *regvalue, UI6 *errcode);
```

この関数はDSPレジスタを読み出しregvalue内の値を戻す。RegIDパラメータはどのレジスタを

む。

【数29】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND

【0033】

【数30】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_NAME_NOT_FOUND

【0034】

【数32】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_MEMPAGE

【0035】

【数34】

【数35】

DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_MEMPAGE

【0036】

【数36】

戻すべきかを指定する。RegIDが-1であれば、全てのレジスタ値が戻される。バッファーを提供了した発呼

者へのポインターである `regvalue` パラメータは、全ての値を保持するのに十分な記憶装置を指示しなければならない。レジスタ ID は特定 DSP 向けであり特定のインプリメンテーションに依存する。`errcode` パラメータは下記の考えられる結果を含む。

【数 37】

DSP_Dbg_Write_Reg

```
BOOL DSP_Dbg_Write_Reg(T_DeviceID DevID, UI6 RegID, T_DSP_Word
                        regvalue, UI6 *errcode);
```

この関数は DSP レジスタに書き込む。Reg ID パラメータはどのレジスタを修正するかを指定する。`regvalue` は書き込む新しい値を含んでいる。レジスタ ID は特定 DSP 向けであり特定のインプリメンテーションに依存する。`errcode` パラメータは下記の考えられる結果を含む。

【数 39】

DSP_Dbg_Set_Break

```
BOOL DSP_Dbg_Set_Break(T_DeviceID DevID, DSP_Ptr addr, UI6 *errcode);
```

この関数は所与のコードアドレス (`addr`) におけるブレイクポイント (`break point`) を設定する。`errcode` パラメータは下記の考えられる結果を含む。

【数 41】

DSP_Dbg_Clr_Break

```
BOOL DSP_Dbg_Clr_Break(T_DeviceID DevID, T_DSP_Ptr addr, UI6
                        *errcode);
```

この関数は所与のコードアドレス (`addr`) において DSP `Dbg_Set_Break` により予め設定されたブレイクポイントをクリアする。`errcode` パラメータは下記の考えられる結果を含む。

【数 43】

```
DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_BP_DID_NOT_EXIST
```

【0040】DSPデバイスドライバ

DSP デバイスドライバ 54 はホスト 12 から DSP 16 への通信を処理する。ドライバ関数はホスト-DSP 通信プロトコルに明記された通信要求を取り上げ、利用可能なハードウェアインターフェイスを介した情報の伝送を処理する。デバイスドライバは RTOS 依存かつ通信ハードウェア依存である。

【0041】DSPライブラリ

DSP ライブラリ 58 は DSP 16 へダウンロードして実行することができるコードのブロックを含んでいる。

```
DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_REGISTER
```

【0037】

【数 38】

```
DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_INVALID_REGISTER
```

【0038】

【数 40】

```
DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
```

【0039】

【数 42】

ダイナミッククロスリンカーが全てのアドレス参照 (`reference`) を決定できるように、コードの各ブロックは予め非リンク、すなわちライブラリとして再配置可能にリンクされている。各コードブロックは DSP

MIPS (百万命令/秒)、優先順位、タイムスライスクオンタム、およびメモリに対するブロックの要求に関する情報も含んでいる。コードブロックヘッダーのフォーマットを表 2 に示す。プログラムメモリおよびデータメモリサイズは、DSP がタスクのメモリ要求をサポートできるかどうかの迅速なチェックをホスト 12 に与えるための近似値である。十分なスペースがあるようであれば、ダイナミッククロスリンカーはコードのリンクおよびロードを試みることができる。ページアライメント (`alignment`) および連続性 (`contiguity`) の要求により、ダイナミッククロスリンカーはそれでも目的を果たせないことがある。好ましい実施例では、コードはバージョン 2 COFF ファイルフォーマットである。

【0042】

【表 2】

表2 コードブロックヘッダー

データタイプ	フィールド名	説明
U16	Processor	The target processor type. (ターゲットプロセッサタイプ)
T_DSP_Name	Name	Task's name. (タスク名)
U32	MIPS	Worst case MIPS required by the task. (タスクが要求する最悪時のMIPS)
T_Size	ProgSize	Total program memory size needed. (所用プログラムメモリサイズ)
T_Size	DataSize	Total data memory size needed. (所用データメモリサイズ)
T_Size	InFrameSize	Size of a frame in the task's input channel. (タスク入力チャネル内のフレームサイズ)
T_Size	OutFrameSize	Size of a frame in the task's output channel. (タスク出力チャネル内のフレームサイズ)
T_Size	InStrmSize	Size of the task's input stream FIFO. (タスクの入力ストリームFIFOのサイズ)
T_Size	OutStrmSize	Size of the task's output stream FIFO. (タスクの出力ストリームFIFOのサイズ)
U16	Priority	Task's priority. (タスクの優先順位)
U32	Quantum	Task's time slice quantum (number of system ticks). (タスクのタイムスライスクォンタム)(システムチック数)
T_Size	StackReq	Stack required. (所用ス택)
T_Size	CoffSize	Total size of the COFF file. (COFFファイルの総サイズ)
T_DSP_Ptr	MsgHandler	Offset to a message handler entry point for the task. (タスクのメッセージハンドラーエントリポイントへのオフセット)
T_DSP_Ptr	Create	Offset to a create entry point that is called when the task is created. (タスク生成時に呼び出される生成エントリポイントへのオフセット)
T_DSP_Ptr	Start	Offset to the start of the task's code. (タスクコードの開始ヘオフセット)
T_DSP_Ptr	Suspend	Offset to a suspend entry point that is called prior to the task being suspended. (保留される(タスク)よりも前に呼び出される保留エントリポイントへのオフセット)
T_DSP_Ptr	Resume	Offset to a resume entry point that is called prior to the task being resumed. (再開される(タスク)よりも前に呼び出される再開エントリポイントへのオフセット)
T_DSP_Ptr	Stop	Offset to a stop entry point that is called prior to the task being deleted. (削除される(タスク)よりも前に呼び出される停止エントリポイントへのオフセット)
T_Host_Ptr	CoffPtr	Pointer to the location of the COFF data in the DSP Library. (DSPライブラリ内のCOFFデータ位置へのポインター)

⑥

【0043】ポータブルコードのリンクされたターゲットコードへの変換

JAVAコード等のポータブル（プロセッサ独立）コードをリンクされたターゲットコードへ変換する手順を図3に示す。この手順は2つの機能、ダイナミッククロスコンパイラ-80およびダイナミッククロスリンクカ-82、を使用する。各機能はホストプロセッサ12上で実現される。好ましい実施例では、ダイナミッククロスリンクカ-80はDSP-APIの一部である。クロスコンパイラ-80はDSP-APIの一部とすることもできる。

【0044】ダイナミッククロスコンパイラ-80はポータブルコードをリンクされない、実行可能なターゲットプロセッサコードへ変換する。ダイナミッククロスリンクカ-82は非リンク、実行可能なターゲットプロセッサコードをリンクされた、実行可能なターゲットプロセッサコードへ変換する。そうするために、DSP16上

ヘローディングする前に、1ブロックのコード内のアドレスを決定しなければならない。ダイナミッククロスリンクカ-82は関数のコードセグメントおよびデータセグメントをリンクし、DSP16上のメモリを割り当て、コードおよび一定のデータをDSP16へロードする。コードを実行するターゲットプロセッサ（すなわち、DSP16）とは異なるプロセッサ（すなわち、ホストプロセッサ12）で関数（コンパイルингおよびリンクイング）が生じるため、この関数は“クロス”コンパイルングおよび“クロス”リンクイングと言われる。

【0045】ダイナミッククロスコンパイラ-80はユーザもしくはユーザエージェント（ブラウザ等）がオンデマンドでロードした予めリンクされていないコードを受け取る。コードは（1）コードの“タグ”部を識別するか、もしくは（2）DSP16で実行する適性について非タグコードセグメントを解析するために処理され

る。ソースコードのタグ部はその中に埋め込まれた “`<start DSP code>`” および “`<end DSP code>`” 等の所定のマークにより DSP へターゲット可能なソースを描くことができる。タグ部が直接もしくは解析により識別される場合には、DSP 16 の現在の処理状態に基づいてクロスコンパイルするかどうかが判断される。コンパイルすると判断されれば、コードのその部分は既知のコンパイラリング方法を使用して非リンク、実行可能ターゲットプロセッサコードを出力するソフトウェアをコンパイラリングすることにより処理される。コンパイルしないという判断は、例えば、DSP 16 により他のタスクが実行されているため、DSP が利用できる容量（一般的に、利用可能 MIPS - 百万命令／秒といわれる）が不十分であるか、利用可能なメモリが不十分である場合になされる。コンパイルされたコードはダイナミッククロスリンカー 82 へ通して DSP 16 において即座に使用するか、あるいは DSP ライブライアリ 58 内に保存することができる。

【0046】ダイナミッククロスリンカー 82 は予め非リンクコードを受け取り、それは（1）ホストプロセッサ 12 と関連してスタティックに格納されるか、（2）ネットワークコネクション（インターネット等のグローバルネットワークを含む）を介してホストプロセッサ 12 へダイナミックにダウンロードされるか、あるいは（3）ダイナミッククロスコンバイラー 80 によりダイナミックに発生される。ダイナミッククロスリンカー 82 はランタイムに決定される DSP 16 のメモリ開始アドレスの入力コードをリンクする。メモリ開始アドレスはホストプロセッサ 12 もしくは DSP 16 により格納され管理されるメモリマップもしくはメモリテーブルから決定することができる。ダイナミッククロスリンカー 82 はコード内の基準メモリ位置を DSP 内の実際のメモリ位置へ変換する。これらのメモリ位置は、例えば、コード内の分岐アドレスもしくはコード内のデータ位置参照（reference）を含むことができる。

【0047】実施例では、ポータブルコードはリンクされているかどうかを含めたコードに関する全情報を含む COFF（共通オブジェクトファイルフォーマット）内にある。リンクされていなければ、コードをリンクするのに変えなければならないアドレスをシンボルテーブルが定義する。

【0048】前記した変換プロセスには従来技術に較べていくつかの著しい利点がある。第 1 に、ダイナミッククロスコンバイラー 80 はダウンロードされたポータブルコードをどこで実行するかについてランタイム判断を行うことができる。例えば、多数のターゲットプロセッサ（2 台の DSP 16 等）を有するシステムでは、ダイナミッククロスコンバイラー 80 は利用可能な資源もしくは能力に基づいてポータブルコードを任意の 1 台のターゲットプロセッサへコンパイルすることができる。ダ

イナミッククロスリンカー 82 はリロケータブルコードをサポートしないターゲットプロセッサでランするようコードをリンクする。コードはランタイムにリンクされるため、DSP 16（もしくは、他のターゲットプロセッサ）内のメモリ位置は保存する必要がなく、デバイス内の全ての計算資源を最適効率で使用することができる。コンパイラリングはプラットホーム 10 のアーキテクチャの知識により達成されるため、一方もしくは両方のプロセッサのインテクションキャッシュアーキテクチャ等の特定プロセッサおよびプラットホーム向けの特徴をコンパイラリングに利用することができる。

【0049】したがって、DSP 16 はその処理能力を完全に使用するようにダイナミックに変えられるさまざまな機能を有することができる。例えば、ユーザは音声認識を含むユーザインターフェイスをロードしたいことがある。その時、ホストプロセッサはソフトウェアをダウンロードして DSP 16 で実行する音声認識ソフトウェアをダイナミックにクロスコンパイルおよびクロスリンクする。あるいは、DSP 16 の現在状態に基づいて DSP ライブライアリ 58 内の予めコンパイルされたソフトウェアをダイナミックにクロスリンクして実行することができる。

【0050】ホストデバイスドライバ
ホストデバイスドライバは DSP 16 からホスト 12 への通信を処理する。ドライバ機能はホスト - DSP 通信プロトコルに明記された通信要求を取り入れて利用可能なハードウェアインターフェイスを介した情報の伝送を処理する。デバイスドライバは RTOS 依存かつ通信ハードウェア依存である。

【0051】ホスト - DSP 通信プロトコル（ホスト - DSP インターフェイスレイヤ）
ホスト - DSP 通信プロトコルはホスト 12 と DSP 16 間のコマンドおよびデータの通信を支配する。通信はいくつかのバス、メッセージ、データチャネル、およびストリーム、からなる。メッセージは初期化パラメータおよびコマンドをタスクへ送るのに使用される。データチャネルはタスク間および DSP 16 とホスト 12 間で大量のデータをデータフレームの形で運ぶ。ストリームはタスク間および DSP 16 とホスト 12 間でストリームとされたデータを通すのに使用される。

【0052】メッセージ
各タスクはメッセージを処理するメッセージハンドラーへのエントリーポイントを有する。メッセージはユーザ定義であり、タスク機能のための初期化パラメータおよびタスクをコントロールするコマンドを含む。タスクはその生成時に指定されるコールバックを介してホスト 12 へメッセージを送る。タスクメッセージハンドラーのプロトタイプおよびホストコールバックのプロトタイプをここに示す。

【数 44】

```
void TaskMsgHandler(T_ReplyRef replyref, T_MsgID MsgID, T_Count count,
T_DSP_Word *buf);
void HostCallBack(T_ReplyRef replyref, T_MsgID MsgID, T_Count count,
T_DSP_Word *buf);
```

【0053】`replyref`パラメータは送信者への返答を返送するのに使用されるインブルメンテーション依存基準値である。各`Send_Message`コールについて、受信者は`replyref`パラメータを使用

<code>Sent message:</code>	<code>MsgPktFlag</code>	<code>taskid</code>	<code>replyref</code>	<code>msgid</code>	<code>count</code>	<code>buf[....]</code>
<code>Reply message:</code>	<code>MsgPktFlag</code>	-1	<code>replyref</code>	<code>msgid</code>	<code>count</code>	<code>buf[....]</code>

マルチワードデータは最下位語が最初に送られる。

【0054】`Send_Message`関数内の0のTaskIDはシステムレベルメッセージを示す。システムレベルメッセージはDSP-API関数を実施するの

Send_Message

```
BOOL Send_Message(T_TaskID TaskID, T_MsgID MsgID, T_Count count,
T_DSP_Word *msgbuf, T_DSP_Word *replybuf, T_Size replybufsize,
T_Count replycount, UI6 *errcode);
```

この関数はTaskIDにより指定されるタスクへユーザ定義メッセージを送る。MsgIDはメッセージを定義し、msgbufは実際のメッセージデータを含んでいる。メッセージサイズはcount T_DSP_Wordsである。メッセージへの返答はreplybufパラメータ内に含まれ、それは呼び出し者により提供されるサイズreplybufsizeのバッファーを指示する。それは特定のメッセージに対する返答を処理するのに十分なサイズでなければならない。errcode

Reply_Message

```
BOOL Reply_Message(T_ReplyRef replyref, T_Count count, T_DSP_Word *buf,
UI6 *errcode);
```

この関数はメッセージへ返答するのに使用される。replyrefはオリジナルメッセージの送信者へ返答を返送するのに使用される参照(reference)であり、特定インブルメンテーション向けである。返答はbufパラメータ内に含まれそのサイズはT_DSP_Wordsである。errcodeパラメータは下記の考えられる結果を含む。

【数49】

```
DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_BAD_REPLY_REF
```

【0057】チャネル

チャネルの概念は1つのプロセッサから別のプロセッサへ、あるいは同じプロセッサ上のタスク間でフレームベースデータを送信するのに使用される。生成されると、

してReply_Messageを呼び出さなければならない。実際のメッセージは次のようにある。

【数45】

に使用される。

【0055】メッセージ関数を下記に示す。

【数46】

eパラメータは下記の考えられる結果を含む。

【数47】

```
DSP_SUCCESS
DSP_DEVID_NOT_FOUND
DSP_DEVID_NOT_RESPONDING
DSP_TASK_NOT_FOUND
```

【0056】

【数48】

チャネルはデータを含むように指定された数およびサイズのフレームを割り当てる。最初に、チャネルは空フレームのリストを含んでいる。データを生じるタスクはデータを書き込む空フレームを要求し、書き込まれるとフレームはチャネルへ戻される。データを消費するタスクはチャネルから完全なフレームを要求し、空になるとフレームはチャネルへ戻される。フレームバッファーのこの要求および戻しにより、最小限のコピーでデータを動かすことができる。

【0058】各タスクは指定された入力および出力チャネルを有する。チャネルが生成されると、それは1つのタスクへの入力、およびもう1つのタスクへの出力として指示される。チャネルのIDにはデバイスIDが含まれ、チャネルはプロセッサ間でデータを通すことができる。ホスト-DSPインターフェイスを横切るチャネルデータフローは下記のようである。

【数50】

ChanPktFlag	Channel ID	Count	Data[...]
-------------	------------	-------	-----------

チャネル関数を下記に示す。

【0059】

Create_Channel

```
BOOL Create_Channel(T_DeviceID DevID, T_Size framesize, T_Count
numframes, T_ChainID *ChainID, UI6 *errcode);
```

この関数はデータフレームベース通信チャネルを生成する。それはカウントおよびサイズが、それぞれ、numframesおよびframesize内に指定されている1組のフレームバッファーのコントロールを維持するチャネルコントロール構造を生成する。生成されると、チャネルはデータフレームを割り当てそれらをその空フレームリストへ加える。ChainIDは新しいチャネルのIDを戻す。DevIDが呼出プロセッサのものでなければ、呼出プロセッサとDevIDプロセッサの両方でチャネルコントロール構造が生成され、通信インターフェイスを横切るデータフローがコントロ

【数51】

ルされる。errcodeパラメータは下記の考えられる結果を含む。

【数52】

```
CHAN_SUCCESS
CHAN_DEVID_NOT_FOUND
CHAN_DEVID_NOT_RESPONDING
CHAN_ALLOCATION_ERROR
```

【0060】

【数53】

この関数はChainIDにより指定される既存のチャネルを削除する。errcodeパラメータは下記の考えられる結果を含む。

【数54】

```
CHAN_SUCCESS
CHAN_DEVID_NOT_FOUND
CHAN_DEVID_NOT_RESPONDING
CHAN_CHANNEL_NOT_FOUND
```

Request_Empty_Frame

```
BOOL Request_Empty_Frame(T_LocalChainID Chn, T_DSP_Word **bufptr,
BOOL WaitFlag, UI6 *errcode);
```

この関数は指定されたローカンチャネルIDから空フレームを要求する。ChnがNULLであれば、タスクの出力チャネルが使用される。戻る時に、bufptrはフレームバッファーへのポインターを含んでいる。WaitFlagがTRUEであり、かつ利用可能なフレームバッファーがなければ、呼出者はバッファーを利用できるようになるまで保留される。WaitFlagがFALSEであれば、機能はとにかく戻る。errcod

eパラメータは下記の考えられる結果を含む。

【数55】

```
CHAN_SUCCESS
CHAN_CHANNEL_NOT_FOUND
CHAN_BUFFER_UNAVAILABLE
```

【0061】

【数56】

Return_Full_Frame

```
BOOL Return_Full_Frame(T_LocalChainID Chn, T_DSP_Word *bufptr, UI6
*errcode);
```

一度タスクがフレームバッファーを満たすと、この関数を使ってチャネルへ戻される。bufptrにより指示されるバッファーは指定されたチャネルIDへ戻され

る。ChnがNULLであれば、タスクの出力チャネルが使用される。errcodeパラメータは下記の考えられる結果を含む。

【数58】

```
CHAN_SUCCESS
CHAN_CHANNEL_NOT_FOUND
CHAN_BUFFER_CTRL_ERROR
```

Request_Full_Frame

```
BOOL Request_Full_Frame(T_LocalChanID Chn, T_DSP_Word **bufptr,
                        BOOL WaitFlag, UI6 *errcode);
```

この関数は指定されたローカンチャネルIDからデータの全フレームを要求する。ChnがNULLであれば、タスクの入力チャネルが使用される。戻る時に、bufptrパラメータはフレームバッファーへのポインターを含んでいる。WaitFlagがTRUEであり、かつ利用可能な完全なフレームバッファーがなければ、呼び出し者はバッファーが利用できるようになるまで保留される。WaitFlagがFALSEであれば、関数

Return_Empty_Frame

```
BOOL Return_Empty_Frame(T_LocalChanID Chn, T_DSP_Word *bufptr, UI6
                        *errcode);
```

タスクはフレームバッファーからのデータを使用したら、この関数を使用してバッファーをチャネルへ戻さなければならない。bufptrにより指示されるバッファーは指定されたチャネルIDへ戻される。ChnがNULLであれば、タスクの入力チャネルが使用される。errcodeパラメータは下記の考えられる結果を含む。

Set_Task_Input_Channel

```
BOOL Set_Task_Input_Channel(T_Task *TaskID, T_ChainID ChanID, UI6
                            *errcode);
```

この関数はタスクの入力チャネルを指定されたチャネルIDに設定する。errcodeパラメータは下記の考えられる結果を含む。

【数64】

```
CHAN_SUCCESS
CHAN_DEVID_NOT_FOUND
CHAN_DEVID_NOT_RESPONDING
CHAN_TASK_NOT_FOUND
CHAN_CHANNEL_NOT_FOUND
```

Set_Task_Output_Channel

```
BOOL Set_Task_Output_Channel(T_Task *TaskID, T_ChainID ChanID, UI6
                            *errcode);
```

この関数はタスクの出力チャネルを指定されたチャネルIDに設定する。errcodeパラメータは下記の考えられる結果を含む。

【数66】

【0063】

【数59】

はとにかく戻る。errcodeパラメータは下記の考えられる結果を含む。

【数60】

```
CHAN_SUCCESS
CHAN_CHANNEL_NOT_FOUND
CHAN_BUFFER_UNAVAILABLE
```

【0064】

【数61】

【数62】

```
CHAN_SUCCESS
CHAN_CHANNEL_NOT_FOUND
CHAN_BUFFER_CTRL_ERROR
```

【0065】

【数63】

【0066】

【数65】

```
CHAN_SUCCESS
CHAN_DEVID_NOT_FOUND
CHAN_DEVID_NOT_RESPONDING
CHAN_TASK_NOT_FOUND
CHAN_CHANNEL_NOT_FOUND
```

【0067】ストリーム

ストリームはフレームに押し入ることはできないが、タスクに対して連続的に流出入するデータのために使用される。ストリームはヘッドおよび流出入時にデータを追跡するテールポインターを付随するサーキュラーバッファ (FIFO) からなっている。各タスクは指定され

StrmPktFlag	Stream ID	Count	Data[...]
-------------	-----------	-------	-----------

ストリーム関数を下記に示す。

た入力および出力ストリームを有することができる。ホスト-DSPインターフェイスを横切るストリームデータフローは次のようにある。

【数67】

【0068】

Create_Stream

```
BOOL Create_Stream(T_DeviceID DevID, T_Size FIFOsize, T_StrmID
*StreamID, UI6 *errcode);
```

この関数は FIFOベース通信ストリームを生成する。それはサイズFIFOsizeのFIFOのコントロールを維持するストリームコントロール構造を生成する。生成されると、ストリームは空FIFOを割り当て、ストリームに流出入するデータフローを処理するようにヘッドおよびテールポインターを初期化する。StreamIDは新しいストリームのIDを戻す。DevIDが呼出プロセッサのものでなければ、呼出プロセッサおよびDevIDプロセッサの両方にストリームコントロール構造が生成され、通信インターフェイスを横切って流

れるデータをコントロールする。errcodeパラメータは下記の考えられる結果を含む。

【数69】

```
STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_ALLOCATION_ERROR
```

【0069】

【数70】

【数70】

この関数はStreamIDにより指定される既存のストリームを削除する。errcodeパラメータは下記の考えられる結果を含む。

【数71】

【0070】

【数71】

```
STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_STREAM_NOT_FOUND
```

Get_Stream_Count

```
BOOL Get_Stream_Count(T_LocalStrmID StrmID, T_Count *count, UI6
*errcode);
```

この関数は現在StrmIDにより指定されるストリームFIFO内にあるT DSP_Wordsのカウントを要求する。countパラメータは戻った時に数を含んでいる。errcodeパラメータは下記の考えられる結果を含む。

【数73】

```
STRM_SUCCESS
STRM_STREAM_NOT_FOUND
```

【0071】

【数74】

Write_Stream

```
BOOL Write_Stream(T_LocalStrmID Strm, T_DSP_Word bufptr, T_Count
count, T_Count countwritten, UI6 *errcode);
```

この関数はStrmにより指定されたストリームへT DSP_Wordsのcount数を書き込む。StrmがNULLであれば、タスクの出力ストリームが使用さ

れる。データはbufptrパラメータにより指示される。戻った時に、countwrittenは実際に書き込まれるT_DSP_Words数を含んでい

る。errcodeパラメータは下記の考えられる結果を含む。

【数75】

```
STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_STREAM_NOT_FOUND
STRM_STREAM_OVERFLOW
```

【0072】
【数76】

Read_Stream

```
BOOL Read_Stream(T_LocalStrmID Strm, T_DSP_Word *bufptr, T_Count maxcount, BOOL WaitFlag, T_Count *countread, UI6 *errcode);
```

この関数はStrmにより指定されたストリームからデータを読み出す。StrmがNULLであれば、タスクの入力ストリームが使用される。データはbufptrにより指定されたバッファー内に格納される。maxcountまでT DSP_Wordsがストリームから読み出される。countreadパラメータは読み出したデータの実際のカウントを含んでいる。errcodeパラメータは下記の考えられる結果を含む。

Set_Task_Input_Stream

```
BOOL Set_Task_Input_Stream(T_Task *TaskID, T_StrmID StrmID, UI6 *errcode);
```

この関数は指定されたストリームIDへタスクの入力ストリームを設定する。errcodeパラメータは下記の考えられる結果を含む。

【数79】

```
STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_TASK_NOT_FOUND
STRM_STREAM_NOT_FOUND
```

【数77】

```
STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_STREAM_NOT_FOUND
```

【0073】
【数78】

【0074】
【数80】

Set_Task_Output_Stream

```
BOOL Set_Task_Output_Stream(T_Task *TaskID, T_StrmID StrmID, UI6 *errcode);
```

この関数は指定されたストリームIDへタスクの出力ストリームを設定する。errcodeパラメータは下記の考えられる結果を含む。

【数81】

```
STRM_SUCCESS
STRM_DEVID_NOT_FOUND
STRM_DEVID_NOT_RESPONDING
STRM_TASK_NOT_FOUND
STRM_STREAM_NOT_FOUND
```

【0075】
【表3】

表3 データタイプ
ここで使用するデータタイプを表3に示す。

シンボル	説明
S8	Signed 8-bit integer. (署名 8 ビット整数)
U8	Unsigned 8-bit integer. (非署名 8 ビット整数)
S16	Signed 16-bit integer. (署名 16 ビット整数)
U16	Unsigned 16-bit integer. (非署名 16 ビット整数)
S32	Signed 32-bit integer. (署名 32 ビット整数)
U32	Unsigned 32-bit integer. (非署名 32 ビット整数)
T_HostWord	A word on the Host processor. (ホストプロセッサ上の語)
T_DSP_Word	A word on the DSP processor. (DSP プロセッサ上の語)
BOOL	Boolean value(TRUE or FALSE). (ブール値(TRUE 又は FALSE)
T_HostPir	Pointer on the Host processor. (ホストプロセッサ上のポインター)
T_DSP_Pir	Pointer on the DSP processor. (DSP プロセッサ上のポインター)
T_DeviceID	Processor device ID. (プロセッサデバイス ID)
T_TaskID	A structure containing fields for a device ID and a processor local task ID. (デバイス ID 及びプロセッサローカルタスク ID 用フィールドを含む構造)
T_ChainID	A structure containing fields for a device ID and a processor local channel ID. (デバイス ID 及びプロセッサローカルチャネル ID 用フィールドを含む構造)
T_MsgID	Message ID. (メッセージ ID)
T_DSP_ID	An object ID on the DSP. (DSP 上のオブジェクト ID)
T_Count	Data type for a count. (カウントのデータタイプ)
T_Size	Data type for a size. (サイズのデータタイプ)
T_HostCallBack	Value used when tasks send message back to the Host. (タスクがホストへメッセージを返送する時の値)
T_ReplyRef	Message reply reference. (メッセージ返答参照)
T_LocalTaskID	Local task ID. (ローカルタスク ID)
T_LocalChanID	Local channel ID. (ローカルチャネル ID)
T_DSP_Name	Name for DSP objects(RTOS dependent). (DSP オブジェクト名 (RTOS 依存))
T_CodeHdr	Code header structure for a DSP Library entry. (DSP ライブラリエントリ用コードヘッダー構造)
T_TaskCreate	Task creation structure. (タスク生成構造)

【0076】システムメッセージ

これらの表はデバイス間を通過するメッセージを定義する（すなわち、ホストから DSP 16 へ）。メッセージをデバイスへ実際にルーティングするのに使用されるため、対応する関数呼出し内にパラメータとして存在するデバイス ID はメッセージには内蔵されない。同様に、

関数呼出しの上半部としてデバイス ID を含むタスク ID はメッセージ内にデバイス ID を含まず、DSP のコードタスク ID 部しか含んでいない。

【0077】

【表4】

表4 DSP-API メッセージ

メッセージ	パラメータ送出	パラメータ返答	方向 Host ⇄ DSP
GET_MIPS	None	U32 mips	→
GET_MEM_AVAIL		T_Size program T_Size datamem	→
ALLOC_MEM	UI16 mempage T_Size size	T_DSP_Word *memptr UI16 errcode	→
FREE_MEM	UI16 mempage T_DSP_Word *memptr	UI16 errcode	→
PUT_BLOB	T_DSP_Ptr destaddr UI16 mempage T_Size size T_DSP_Word BLOB[size]	UI16 errcode	→
CREATE_TASK	T_TaskCreate tcs	T_TaskID TaskID UI16 errcode	→
START_TASK	T_TaskID TaskID	UI16 errcode	→
SUSPEND_TASK	T_TaskID TaskID	UI16 errcode	→
RESUME_TASK	T_TaskID TaskID	UI16 errcode	→
DELETE_TASK	T_TaskID TaskID	UI16 errcode	→
CHANGE_PRIORITY	T_TaskID TaskID UI16 newpriority	UI16 oldpriority UI16 errcode	→
GET_TASK_STATUS	T_TaskID TaskID	UI16 status UI16 priority T_ChainID Input T_ChainID Output UI16 errcode	→
GET_ID	T_DSP_Name Name	T_DSP_ID ID UI16 errcode	→

【0078】

【表5】

表5 インターフェイスレイヤ/チャネルインターフェイスレイヤメッセージ

メッセージ	パラメータ送出	パラメータ返答	方向 Host ⇄ DSP
CREATE_CHANNEL	T_Size framsize T_Count numframes	T_ChannelID ChannelID UI16 errcode	→
DELETE_CHANNEL	T_ChannelID ChannelID	UI16 errcode	→
CREATE_STREAM	T_Size FIFOsize	T_StreamID StreamID UI16 errcode	→
DELETE_STREAM	T_StreamID StreamID	UI16 errcode	→

【0079】

【表6】

表6 デバッグメッセージ

メッセージ	パラメータ送出	パラメータ返答	方向 Host ⇄ DSP
READ_MEM	UI16 mempage T_DSP_Ptr addr T_Count count	T_DSP_Word mem[count] UI16 errcode	→
WRITE_MEM	UI16 mempage T_DSP_Ptr addr	UI16 errcode	→
READ_REG	T_Count count T_DSP_Word mem[count]		
WRITE_REG	UI16 RegID T_DSP_Word regvalue	DSP_WORD regvalue UI16 errcode	→
SET_BREAK	T_DSP_Ptr addr	UI16 errcode	→
CLR_BREAK	T_DSP_Ptr addr	UI16 errcode	→
BREAK_HIT	T_DSP_Ptr addr	UI16 ACK	←

【0080】ダウンロードィングネーティブコード

図4-図6はネーティブコードをターゲットプロセッサ(すなわち、ホスト12もしくはDSP16)へ確実かつ効率的にダウンロードする実施例を示す。コードをダウンロードするこの実施例は、例えば、インターネット、もしくは他のグローバルネットワーク、ローカルもしくはワイドエリアネットワーク、もしくはPCカードやスマートカード等の周辺装置からコードをダウンロードするのに使用できる。

【0081】図4にJAVA Bean90の実施例を示し、Bean90はネーティブコード92のラッパー

(wrapper)として作用する。BeanはさらにCode Type属性94a、Code Size属性94bおよびMISP要求属性94cとして表記されているいくつかの属性94を含んでいる。Bean90はLoad Codeアクション96a、Load Parametersアクション96bおよびExecute Parameter96cを含むいくつかのアクション96を有している。

【0082】動作において、Load Codeアクション96aは外部ネーティブコード(ターゲットプロセッサにとってネーティブ)をBean内へロードするの

に使用される。JAVA Beansはパーシステンス(persistence)を有するため、Bean90はネーティブコード92および属性94を含むその内部状態を格納することができる。Load Parametersアクション96bはネーティブコード92から(例えば、前記したCOFFファイルフォーマットを使用して)パラメータを検索し属性94a-cとして格納する。Executeアクション96cはDSP16内にインストールされたタスクを実行する。

【0083】Bean90を使用してコードをターゲットプロセッサへダウンロードする様子を図5に示す。この例では、ターゲットプロセッサはDSP16(もしくは、多数のDSP16の中の1つ)であるものとするが、それはネーティブコードをホストプロセッサ12へダウンロードするのにも使用できる。さらに、所望のBean90はLANサーバもしくはインターネットサーバ等のネットワークサーバ内に常駐しているものとすると、Beanはスマートカード等のプラットホーム10と通信する任意のデバイス内に常駐することができる。ワイヤレスデータプラットホーム10については、ネットワークサーバ100への接続はしばしばワイヤレスである。

【0084】図5において、プラットホーム10はネットワークサーバ100に接続されている。図2に詳細に示すホストプロセッサ12はJAVAバーチャルマシン44を介して1つ以上のJAVAアプレット40を実行することができる。新しいコードをダウンロードするために、ホスト12はネットワークサーバ100からBean90を含むアプレットをロードするか、もしくはアプレットを含まないBeanをサーバ100からダウンロードすることができる。ラッパーBean90が検索されると、ネーティブコードのサイズ、コードタイプ(コードはどのプロセッサ用か)および必要なMIPSを質問することができる。所期のプロセッサがコード92を実行するのに十分な資源を有する場合には、コード92は所期のプロセッサ、図5に示すアキテクチャ内のホストプロセッサ12もしくはDSP16、で実行するようにインストールすることができる。典型的には、ネーティブコード92は非リンク、コンパイルされたコードである。したがって、DSP-API50のクロスリンカー82はコードを利用可能なメモリ位置へリンクする。Beanは2進ネーティブコード92をダイナミッククロスリンカー82へ通し、それがコードをインストールして実行する。

【0085】典型的に、ネーティブコードのダウンロードが生じるのは、その中にDSP関数が所望されるアプレットをユーザが実行している場合である。最初に、アプレットは所望のコードがタスク60としてDSP内にインストールされている、またはDSPライブラリ58内で利用可能であるかをチェックする。もしそうならタ

スクはダウンロードなしで実行できる。

【0086】タスクがDSP16もしくはDSPライブラリ58内に格納されていなければ、オブジェクト(ここでは、“DSPLoader”オブジェクトと言う)を生成してBean90をロードすることができる。DSPLoaderクラスがホスト12上でローカルであれば、JAVAはBeanもローカルに利用可能であるかをチェックする。最初の場合、ローカルに格納されたコードを有するBeanがあることがある。そうであれば、BeanからのコードはDSP(もしくは、Code Typeにより指定されるいずれかのプロセッサ)へインストールされる。コードのないBeanがローカルに格納される場合には、Beanは適切なサーバからコードを検索することができる。

【0087】一方、DSPLoaderオブジェクトがローカルではなければ、JAVAはアプレットを書き込んだサーバからBean90をロードする。次に、Beanからのコードは前記したようにインストールされる。

【0088】JAVA Beanの使用に関連してネーティブコードのダウンローディングが説明されるが、ActiveXアプレット等の他の言語内にコードを包み込んで達成することもできる。

【0089】JAVA Bean(もしくは、他のアプレット)をネーティブコードのラッパーとして使用することは著しい利点がある。最初に、複数のプロセッサの中の1つへコードをローディングするに単純な標準的方法でよい。Beanが生成され、コードがBeanへロードされて適切なプロセッサとリンクされる。Bean内にコードをラッピングすることなく、プロセスは数百のステップをとることができる。第2に、多数のネーティブコード片を1つのアプレットにより結合し、1つのアプレットを使用する多数の個別のルーチンから複雑なアプリケーションを発生してルーチンを所望により結合することができる。第3に、言語のセキュリティ機能を利用ることができ、Bean90内のJAVAコードだけでなくネーティブコード92も保護される。ActiveX等の他の言語もセキュリティ機能を有する。

【0090】セキュリティ

2つの最重要セキュリティ機能はデジタル署名および暗号化である。JAVA BeanもしくはActiveXアプレットはコードソースにより署名することができ、Beanもしくはアプレットがダウンロードされると、署名は信頼できるソースのリストを有する受信アプリケーションにより照合される。Beanもしくはアプレットに信頼できるソースの署名があれば、標準技術を使用して解読することができる。したがって、ネーティブコードは伝送中にBeanもしくはアプレットのコードと共に暗号化され、コードの非認可修正が防止される。ネーティブコードは安全であり信頼できるソースから来る

ため、属性も正確であるものと信頼することができる。

【0091】図6はJAVA Beanを使用するプロセッサのネーティブコードダウンロードイングプロセスを説明するフロー図であり、ネーティブコードは同様な技術を使用して異なる言語のアプレットで包むことができることがお判りであろう。ステップ110において、暗号化されたデジタル署名Bean90はJAVAバーチャルマシンを動かすデバイスへダウンロードされる。ステップ112において、署名が照合される。信頼できるソースとして表記されたソースからのものでなければ、ステップ114において例外処理がイネーブル(enable)される。Beanが信頼できるソースから来るものである場合、ユーザがそのソースで不自由でなければ、例外処理関数はユーザにBeanを受け入れる機会を与えることができる。署名が無効であれば、例外処理はBean90を削除してユーザへ適切なエラーメッセージを送ることができる。

```

package ti.dsp.loader;

import java.awt.*;
import java.io.*;
import java.net.*;

public class NativeBean extends Canvas implements Serializable
{
    public NativeBean() {
        setBackground(Color.white);
        funcData = new ByteArrayOutputStream();
        try {
            funcCodeBase = new URL("http://localhost");
        }
        catch (MalformedURLException e) {
    
```

【表8】

【0092】署名が有効で信頼できるソースから来るものであれば、ステップ116においてBeanが解読される。このステップではJAVAコードとBean内のネーティブコードの両方が解読される。ステップ118において、Bean90から属性が検索されステップ120において、アプレットは適切なプロセッサがコードを実行するのに十分な資源を有するかどうかを確認する。十分な資源を有しない場合には、例外処理ステップ114はネーティブコードのインストールを拒絶することができ、あるいは資源を解放するステップをとることができます。十分な資源がある場合には、ステップ122においてコードはクロスリンカーを使用してリンクされ所望のプロセッサにインストールされる。ステップ124において、ネーティブコードが実行される。

【0093】Bean90のサンプルJAVAスクリプトを下記に示す。

【表7】

```

        }

    public Dimension getMinimumSize() {
        return new Dimension(50, 50);
    }

    public void loadCode() {
        URL baseURL = null;

        try {
            baseURL = new URL(funcCodeBase.toString() + "/" + myFunction);
        }
        catch (MalformedURLException e) {
        }

        DataInputStream source = null;
        int read;
        byte[] buffer;

        buffer = new byte[1024];
        try {
            source = new DataInputStream(baseURL.openStream());
        }
        catch (IOException e) {
            System.out.println("IOException creating streams: " + e);
        }

        codeSize = 0;

        funcData.reset();

        try {
            while (true) {
                read = source.read(buffer);

                if (read == -1)
                    break;

                funcData.write(buffer, 0, read);
            }
        }
        catch (IOException e) {
            System.out.println("IOException: " + e);
        }
    }
}

```

【表9】

```

codeSize = funcData.size();
System.out.println("Code size = " + codeSize);

try {
    source.close();
}
catch (IOException e) {
    System.out.println("IOException closing: " + e);
}

public synchronized String getFunctionName() {
    return myFunction;
}

public void setFunctionName(String function) {
    myFunction = function;
}

public synchronized String getCodeBase() {
    return funcCodeBase.toString();
}

public void setCodeBase(String newBase) {
    try {
        funcCodeBase = new URL(newBase);
    }
    catch (MalformedURLException e) {
    }
}

public void installCode() {
    FileOutputStream destination = null;
    File libFile = new File(myFunction);

    try {
        destination = new FileOutputStream(libFile);
    }
    catch (IOException e) {
        System.out.println("IOException creating streams: " + e);
    }

    if (destination != null) {
}

```

【表10】

```

        try {
            funcData.writeTo(destination);
        }
        catch (IOException e) {
            System.out.println("IO Exception installing native code: " + e);
        }
    }
}

linkCode(funcData)

public void loadParameters() {
}

public void execute() {
}

public synchronized int getCodeSize() {
    return codeSize;
}

public synchronized int getCodeType() {
    return codeType;
}

public void setCodeType(int newType) {
    codeType = newType;
}

private int codeSize = 0;
private int codeType = 1;
private String myFunction = "";
private URL funcCodeBase = null;
private ByteArrayOutputStream funcData = null;
}
}

```

【0094】前記したスクリプトにおいて、NativeBean() ルーチンはネイティブコードを保持する Bean90 を生成する。loadCode() ルーチンはサーバからネイティブコードを得る。getFunctionalName() および getCodeBase() ルーチンは属性を検索する。installCode() ルーチンはクロスリンカーを呼び出してネイティブコードを DSP にリンクしリンクしたコードをコードする。loadParameters() ルーチンはネイティブコードを調べてその属性を確認するよう Bean を命令する。getCodeSize() および getCodeType() ルーチンは属性を要求するアプレットへ転送する。

【0095】本発明の詳細説明はある代表的な実施例に向けられたが、当業者ならば別の実施例だけでなくこれらの実施例のさまざまな修正を考えられるであろう。請

求の範囲に入る修正および別の実施例は全て本発明に含まれるものとする。

【0096】以上の説明に関して更に以下の項を開示する。

(1) 移動電子装置であって、ネイティブコードを実行するコプロセッサと、ホストプロセッサシステムおよびプロセッサ独立コードに対応するネイティブコードを実行するように作動するホストプロセッサシステムであつて、デジタル信号コプロセッサにより実施されるタスクをダイナミックに変化させるように作動するホストプロセッサシステムと、前記ホストプロセッサシステムと前記コプロセッサ間の通信回路と、を含む移動電子装置。

【0097】(2) 第1項記載の移動電子装置であつて、前記コプロセッサはデジタル信号処理装置である、移動電子装置。

【0098】(3) 前記いずれか1項記載の移動電子装

置であって、前記プロセッサ独立コードはJAVAを含む、移動電子装置。

【0099】(4) 前記いずれか1項記載の移動電子装置であって、前記ホストプロセッサシステムは前記コプロセッサのネーティブコードを発生することができる、移動電子装置。

【0100】(5) 前記いずれか1項記載の移動電子装置であって、前記ホストプロセッサシステムはプロセッサ独立ソースコードをコンパイルすることにより前記コプロセッサのネーティブコードを発生することができる、移動電子装置。

【0101】(6) 前記いずれか1項記載の移動電子装置であって、前記ホストプロセッサシステムはソースコードの識別されたブロックをコンパイルする、移動電子装置。

【0102】(7) 前記いずれか1項記載の移動電子装置であって、前記ホストプロセッサシステムはコプロセッサで実行できるソースコードのブロックを識別してコードの前記ブロックをコンパイルする、移動電子装置。

【0103】(8) 前記いずれか1項記載の移動電子装置であって、さらに前記コプロセッサへダウンロードして実行することができるルーチンのライブラリを格納するメモリを含む、移動電子装置。

【0104】(9) 前記いずれか1項記載の移動電子装置であって、さらにハードウェア言語アクセラレータを含む、移動電子装置。

【0105】(10) 前記いずれか1項記載の移動電子装置であって、前記ハードウェアアクセラレータはJAVAアクセラレータを含む、移動電子装置。

【0106】(11) 第1項記載の移動電子装置であって、さらにネットワークからデータを受信するネットワークインターフェイス回路を含む、移動電子装置。

【0107】(12) 移動電子装置の制御方法であって、コプロセッサにおいてネイティブコードを実行するステップと、ホストプロセッサシステムにおいてネイティブコードおよびプロセッサ独立コードを実行するステップと、デジタル信号コプロセッサにより実施されるタスクを前記ホストプロセッサシステムによりダイナミックに変えるステップと、前記ホストプロセッサシステムと前記コプロセッサ間で通信を行うステップと、を含む方法。

【0108】(13) 第12項記載の方法であって、コプロセッサにおいてネイティブコードを実行する前記ステップは、デジタル信号処理装置においてネイティブコードを実行するステップを含む、方法。

【0109】(14) 第12項、第13項記載の方法であって、さらに前記一般的処理システムにおいて前記コプロセッサのネイティブコードを発生するステップを含む、方法。

【0110】(15) 第14項記載の方法であって、ネ

イティブコードを発生する前記ステップはプロセッサ独立ソースコードをコンパイラリングしてネーティブコードを発生するステップを含む、方法。

【0111】(16) 第15項記載の方法であって、さらに前記ソースコードのブロックを識別して前記コプロセッサで実行するようコンパイルするステップを含む、方法。

【0112】(17) 第12項ー第16項記載の方法であって、さらに前記ホストプロセッサシステムから前記コプロセッサへダウンロードして実行するルーチンのライブラリを格納するステップを含む、方法。

【0113】(18) 移動電子装置であって、複数個のコプロセッサと、ホストプロセッサシステムであって、ソースコードを実行し、前記1個以上のコプロセッサで実行されるソースコードの1つ以上のセクションを識別し、ソースコードの識別した各セクションについて、対応するコプロセッサを決定し、ソースコードの識別した各セクションについて、コードの前記識別したセクションを前記対応するコプロセッサに関連するネーティブコードへコンパイルして前記対応するコプロセッサへインストールする、ように作動する前記ホストプロセッサシステムと、前記ホストプロセッサシステムと前記コプロセッサ間の通信回路と、を含む、移動電子装置。

【0114】(19) 移動電子装置の制御方法であって、ホストプロセッサシステムでソースコードを実行するステップと、1個以上のコプロセッサで実行されるソースコードの1つ以上のセクションを識別するステップと、ソースコードの識別した各セクションについて、対応するコプロセッサを決定するステップと、ソースコードの識別した各セクションについて、コードの前記識別したセクションを前記対応するコプロセッサに関連するネーティブコードへコンパイラリングして前記ネーティブコードを前記対応するコプロセッサへインストールするステップと、前記ホストプロセッサシステムと前記コプロセッサ間の通信を行うステップと、を含む、方法。

【0115】(20) 複数個のプロセッサ(12, 16)を含むワイヤレスデータプラットホーム(10)。タスクが実施される時に情報を通信できるようにプロセッサ間に通信チャネルが開設される。1個のプロセッサで実施されるダイナミッククロスコンパイラ(80)がコードを別のプロセッサのためのネーティブ処理コードへコンパイルする。ダイナミッククロスリンク(82)が他のプロセッサのためにコンパイルされたコードをリンクする。ネーティブコードはそれを包むJAVA Bean(90)(もしくは他の言語タイプ)を使用してプラットホームへダウンロードすることもできる。JAVA Beanはセキュリティのために暗号化しデジタル署名することができる。

【0116】関連出願の相互参照
本出願は同じ日付で出願され本開示の一部としてここに

組み入れられている、ウースリー等の米国特許出願第08/995,600号“Mobile Communication System with Cross Compiler and Cross Linker”(アットニードケット第26453号)、ブリュワーの米国特許出願第08/995,597号“Method and Apparatus for Providing Downloadable Functionality to an Embedded C oprocessor”(アットニードケット第26440号)、およびブリュワーの米国特許出願第08/995,603号“Method and Apparatus for Extending Security Model to Native Code”(アットニードケット第26439号)に関連している。

【図面の簡単な説明】

【図1】一般的なワイヤレスデータ処理に特に適したプラットホームアーキテクチャのブロック図。

【図2】図1のプラットホームの機能ブロック図。

【図3】ダイナミッククロスコンパイルおよびダイナミッククロスリンク機能の機能ブロック図。

【図4】デバイスへダウンロードするJAVA Beanラッパー内に包まれたプロセッサで実行するネーティブコードの実施例。

【図5】遠隔サーバ上に配置されたJAVA Beanからデバイス上のプロセッサへ包まれたネーティブコードを転送する操作を示す図。

【図6】図5の操作に関するセキュリティ機能を記述するフロー図。

【符号の説明】

10 ワイヤレスデータプラットホーム

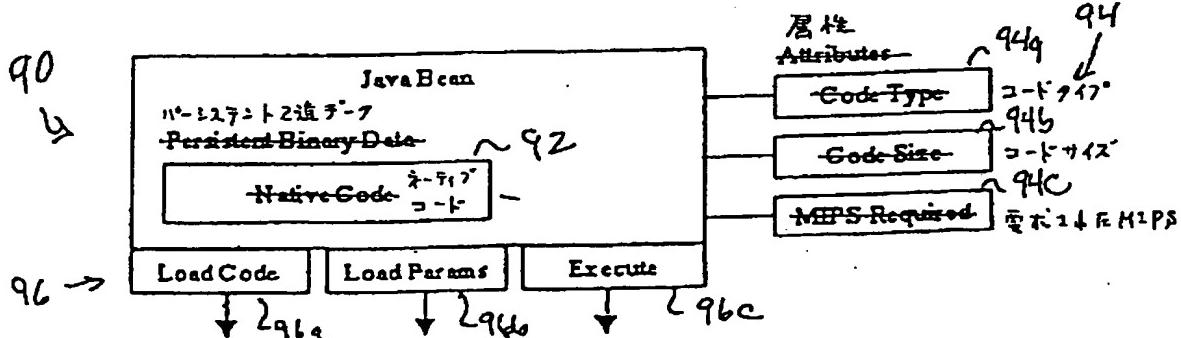
12 汎用ホストプロセッサ

14 パス構造

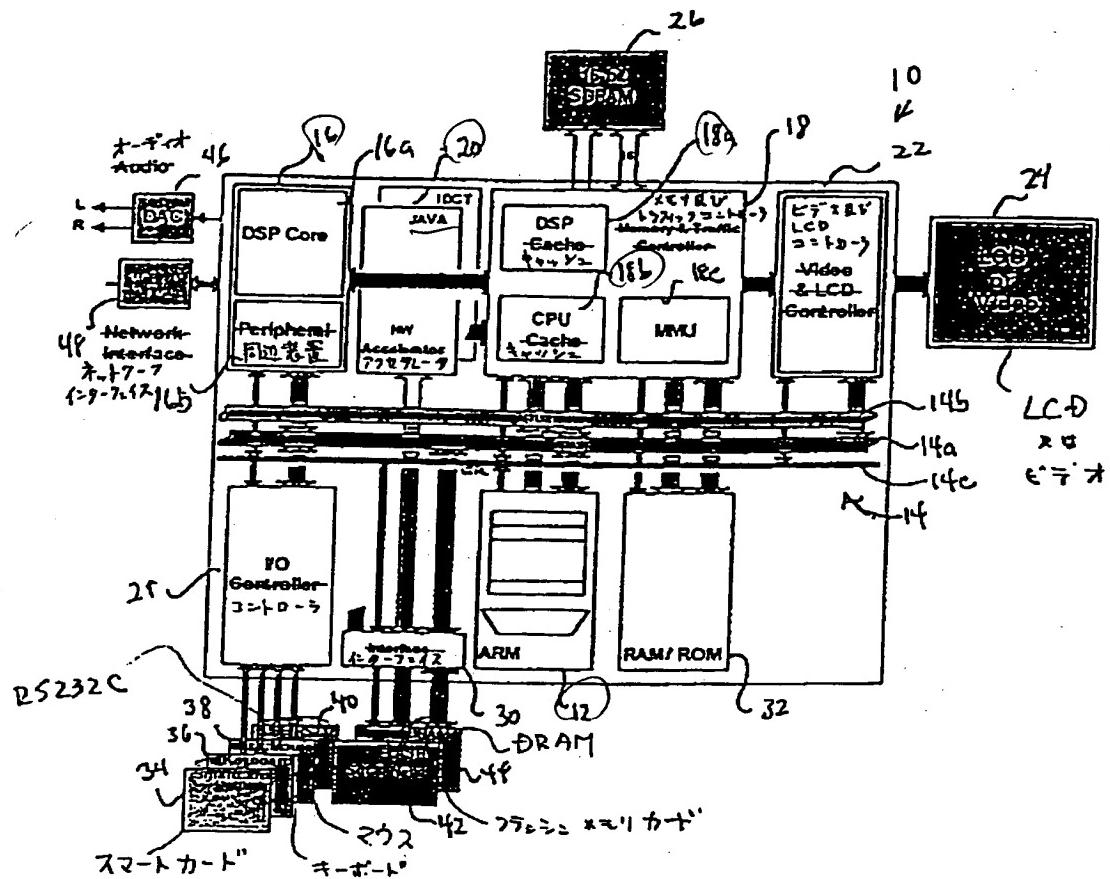
16 DPS (デジタル信号処理装置)

- 18 トラフィックコントローラ
- 20 ハードウェアアクセラレータ回路
- 22 ビデオおよびLCDコントローラ
- 24 LCDもしくはビデオディスプレイ
- 26 主記憶装置
- 28 I/Oコントローラ
- 30 インターフェイス
- 32 RAM/ROM
- 34 スマートカード
- 36 キーボード
- 38 マウス
- 40 シリアルポート
- 42 フラッシュメモリカード
- 44 DRAMカード
- 46 DAC (デジタル/アナログコンバータ)
- 48 ネットワークインターフェイス
- 50 DSP API (アプリケーションプログラミングインターフェイス)
- 52 ホストDSPインターフェイスレイヤ
- 54 DSPデバイスドライバ
- 56 ホストRTOS (リアルタイムオペレーティングシステム)
- 58 DSPライブラリ
- 60 タスク
- 62 ホスト-DSPインターフェイスレイヤ
- 64 DRTOS
- 66 ホストデバイスドライバ
- 70 ホスト-DSPインターフェイス
- 80 ダイナミッククロスコンパイラ
- 82 ダイナミッククロスリンク
- 90 JAVA Bean
- 92 ネーティブコード
- 94 属性
- 96 アクション
- 100 ネットワークサーバ

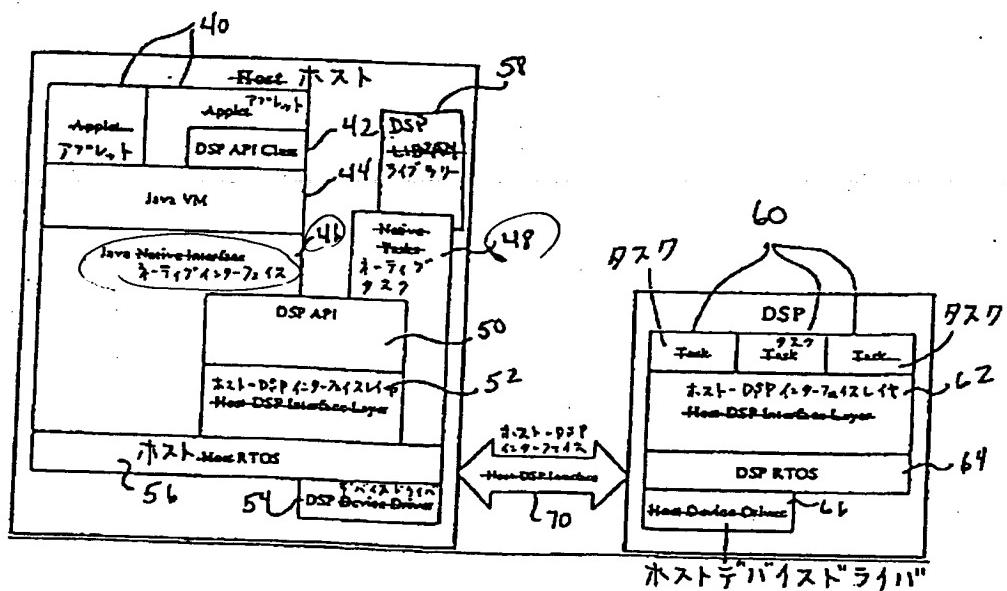
【図4】



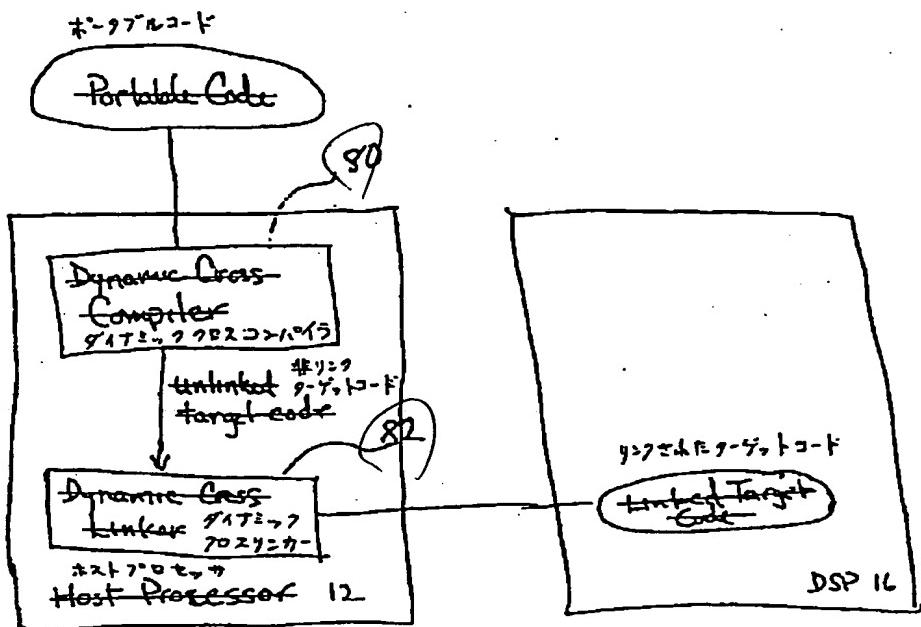
【図1】



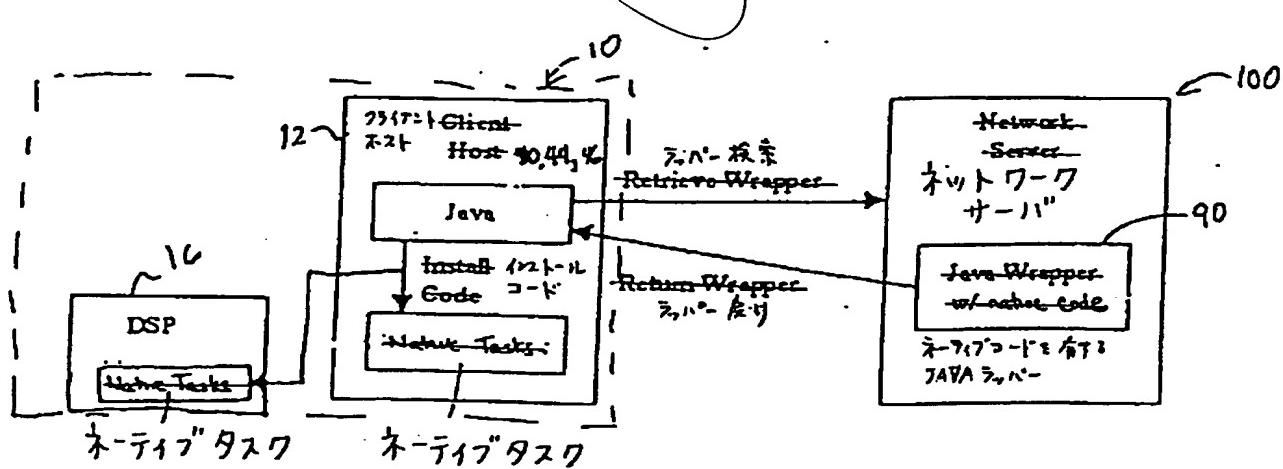
【図2】



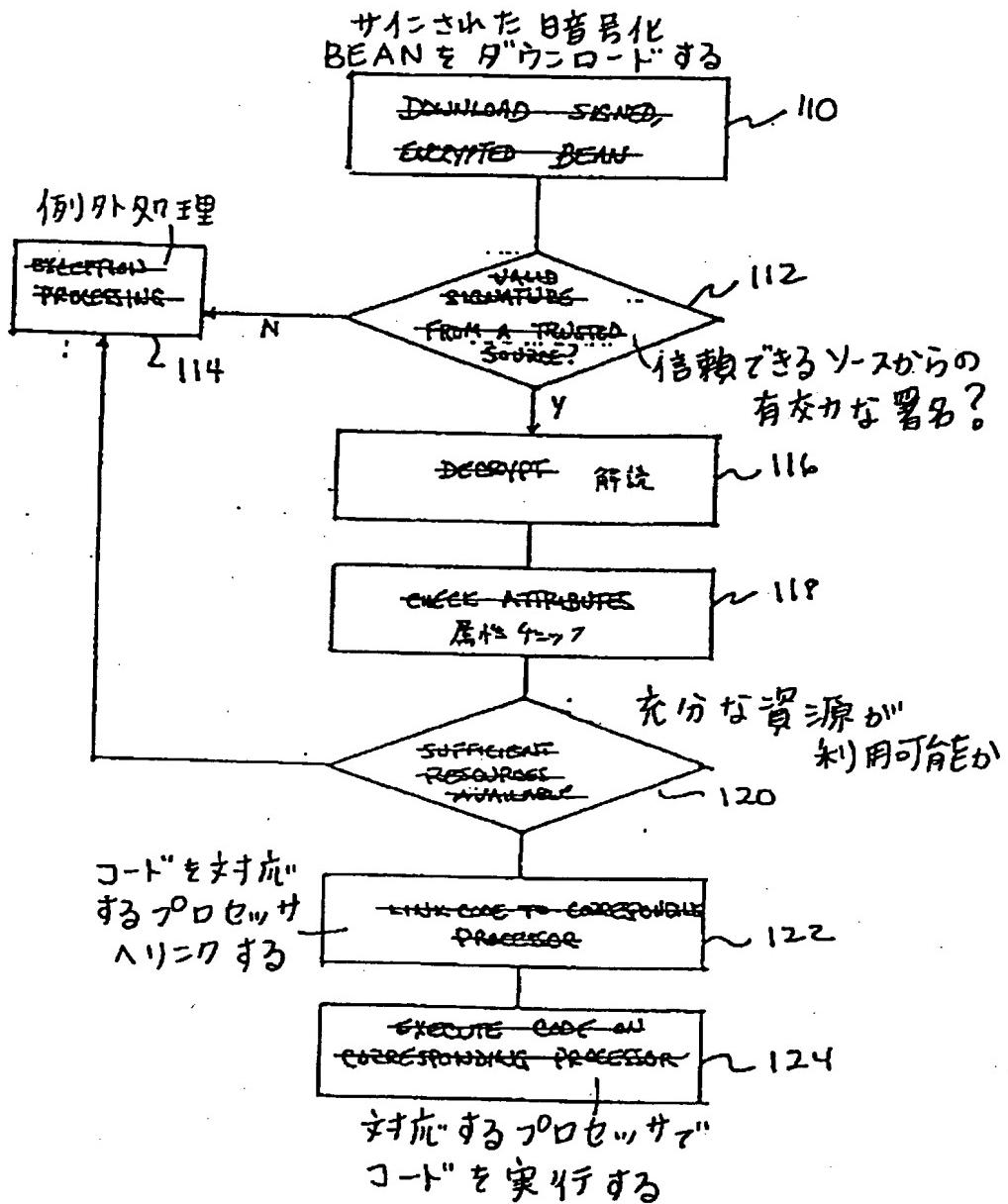
【図3】



【図5】



【図6】



フロントページの続き

(72)発明者 マシュー エイ. ウールゼイ
アメリカ合衆国 テキサス州ブラン, ミド
ルコープ 608

(72)発明者 ジエラール ショベル
フランス国 アンティーブ, シュマンド
ラ シュケット, レ ベルジェルド
バル コンスタンス